

Modeling Safe Reconfiguration with the FUJABA Real-Time Tool Suite ^{*}

Claudia Priesterjahn, Matthias Tichy
Software Engineering Group
University of Paderborn
Warburger Str. 100
D-33098 Paderborn, Germany
[cpr|mtt]@uni-paderborn.de

ABSTRACT

Software systems are increasingly built to exhibit self-* properties (e.g. self healing or self optimization) which require reconfiguration and change at runtime. This is even true for embedded or mechatronic systems which are often used in safety critical environments. In those cases, the effects of the reconfiguration on the safety of the system must be carefully analyzed. We present an approach to ensure the safety of self-* systems during runtime by checking whether a reconfiguration is allowed w.r.t. the hazard probability and the associated damage after the reconfiguration. The approach has been implemented as plugins for the Fujaba Real-Time Tool Suite.

1. INTRODUCTION

Embedded systems are often used in a safety-critical context. Consequently, hazards and risks have to be considered during system development. Standard development approaches (cf. [Lev95]) for safety-critical computer systems require hazards to be identified and the associated damage to be defined. The damage is the result when a hazard leads to an accident. We define risk as the product of the hazard probability and the damage of the accident linked to the hazard. We refer to [Lev95, Sto96] for more detailed definitions.

Self-* systems pose a challenge to these activities as they change their behavior and structure during runtime. This leads to changes of the hazard probabilities and damage values, which in turn affect the associated risks. However, this also opens up possibilities for risk management as it enables to adapt the behavior, and thus the hazard probability, by reacting to changes in the damage during runtime. Consequently instead of checking whether the *worst case damage* results in a safe risk for all configurations before runtime, we check during runtime whether the target configuration of a reconfiguration activity is safe w.r.t. the current damage.

A recent approach on risk analysis computes risk component wise and combines the results for a static component structure [YA02]. In contrast to our work, Yacoub and Ammar do not address self-* systems with different configurations during runtime.

^{*}This work was developed in the course of the Collaborative Research Center 614 – Self-optimizing Concepts and Structures in Mechanical Engineering – University of Paderborn, and was published on its behalf and funded by the Deutsche Forschungsgemeinschaft.

In previous works [GTS04, GT06, THMvD08], we have presented a component-based hazard analysis approach which also considers the associated risk. This approach is specifically tailored to self-* systems which change their behavior by structural adaptation.

The approach extends all components by an abstract failure propagation which relates random errors in the components to failures at the components' ports. We follow the terminology of Laprie [Lap92] by associating *failures* – the external visible deviation from the correct behavior – to the ports where the components interact with their environment. *Errors* – the manifestation of a *fault* in the state of a component – are restricted to the internal of the component. We use Boolean logic with quantifiers to formally encode the failure propagation of the system and the occurrence of hazards.

The failure propagation models of all components in the system structure are then combined to form the system failure propagation. This system failure propagation is analyzed w.r.t. the errors which have to manifest so that a given hazard occurs. The hazard probability is computed based on the individual probabilities of the errors. Finally, the risk is the product of hazard probability and damage.

For the special case of self-* systems which change their structure during runtime, the approach supports to compute all structural configurations. These configurations are also considered in the aforementioned analysis. We determine the configurations in which a hazard can occur and, quantitatively, the configurations with the highest and worst hazard probability and risk. We currently pessimistically abstract from different damage values during runtime, which would result in different risk values, by employing the worst case damage.

In this paper, we present how we deal with damage values which change during runtime. The basic idea is that we refine the structural adaptation behavior of the system components by additional checks whether the reconfiguration to another structure is allowed with respect to the current damage value and the defined maximum risk.

In our approach, the reconfiguration behavior is distributed over the system components but the hazard probability depends on the system structure which is typically not known

by the individual components. Therefore, we opted to add a central component called the Risk Coordinator. This component is asked by a component whether a reconfiguration is safe with respect to the risk and allows or disallows this reconfiguration.

We exemplify our approach based on a scenario from our real-life RailCab project. The RailCab project¹ was founded at the University of Paderborn in 1998 in order to develop a new railway system which features the advantages of both public and individual transport in terms of cost and fuel efficiency as well as flexibility and comfort. The novel system is characterized by autonomous vehicles operating on demand instead of trains being determined to a fixed schedule. One particular goal of the RailCab project is to reduce the energy consumption due to air resistance by coordinating the autonomously operating RailCabs in such a way that they build convoys whenever possible. The RailCabs use different configurations w.r.t. whether they are driving alone or in a convoy. In the first case, they use a speed controller whereas they use a sophisticated distance controller in the second case. The damage associated to a hazard can change during runtime, e.g. in response to loading and unloading of goods like hazardous materials.

We present our approach in the following section. This presentation focuses on the behavioral extension of the reconfiguration behavior taking the real-time characteristics of embedded systems into account. In Section 3 we sketch how the presented approach has been implemented in the Fujaba4Eclipse Real-Time Tool Suite. Finally, we give a conclusion and an outlook on future work.

2. SATISFYING RISK CONSTRAINTS DURING RECONFIGURATION

In order to avoid configurations that exceed the maximum risk we enhance our system by a central component that performs risk analysis for the requested configuration to allow or prohibit reconfiguration. Therefore, we first introduce the extension of the architecture before we present the extension of the behavior models that is necessary to enable the blocking of reconfiguration transitions.

2.1 Architecture Extension

The system architecture is extended by an additional component which supervises the reconfigurations of the system, namely the *Risk Coordinator*. Before executing a reconfiguration each component sends a request to the Risk Coordinator. The Risk Coordinator computes the risk of the total system after the requested reconfiguration and checks if the system would still satisfy the maximum risk level after this reconfiguration. Depending on this verdict the Risk Coordinator allows or prohibits the reconfiguration. Further, each system component contains a sub component *Sub Coordinator* that encapsulates the communication between this component and the Risk Coordinator. Figure 1(a) shows an exemplary system consisting of two RailCabs. Figure 1(b) shows the same architecture extended by a Risk Coordinator instance and a Sub Coordinator instance for each RailCab instance.

¹<http://www-nbp.upb.de/en/index.html>

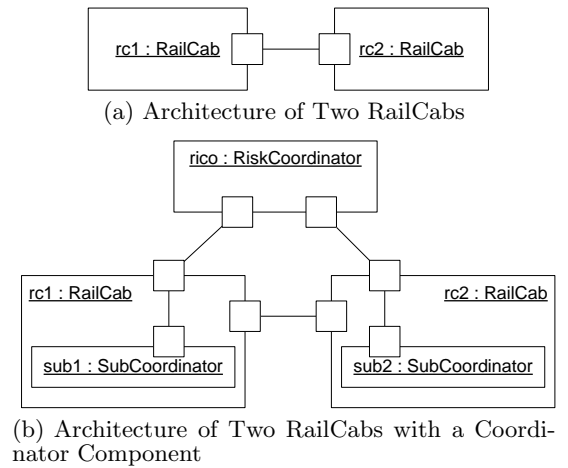


Figure 1: Architecture Extension

2.2 Behavior Extension

We define the component's behavior by hybrid Reconfiguration Charts - UML state machines extended by time, continuous behavior and reconfiguration. Reconfigurations are specified by embedding configurations into the states. A reconfiguration can result in a different hazard probability for the system or a different damage value.

We introduce *Safety Transitions* as an extension of hybrid Reconfiguration Charts. Safety Transitions can be blocked in order to prevent unsafe configurations. Figure 2 shows an example of a Safety Transition between the states *noConvoy* and *convoy* representing the reconfiguration taken in order to join or build a convoy of RailCabs. Safety Transitions are drawn as a fat line with the label $\llcorner\text{Safety Transition}\gg$.

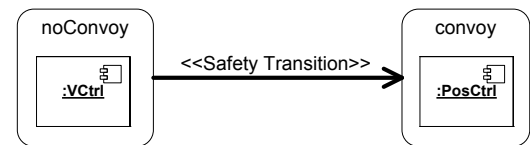


Figure 2: Safety Transition

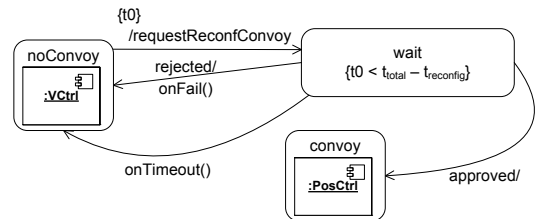


Figure 3: Safety Transition Semantics

The semantics of the Safety Transition are shown in Figure 3. We add the state *wait* between the states *noConvoy* and *convoy* connected by the Safety Transition. The reconfiguration request to the Risk Coordinator is triggered by the message *RequestReconfConvoy* of the transition (*noConvoy*, *wait*). In case the Risk Coordinator answers

with *approved* the reconfiguration is executed and the system switches to *convoy*. If the Risk Coordinator rejects the request, the system switches back to *noConvoy* and executes the side effect *onFail()*. *onFail()* is a method provided by the developer in order to react to the rejection. In our example the RailCab would inform the RailCab driving behind about the rejection and brake the convoy.

To guarantee a WCET for a Safety Transition, we add the clock $t0$ and a timing constraint: the invariant $\{t0 < t_{total} - t_{reconfig}\}$ for the state *wait*. t_{total} represents the time needed for the total Safety Transition. $t_{reconfig}$ names the time needed for the reconfiguration itself, namely the transition (*wait, convoy*). Consequently, $t_{total} - t_{reconfig}$ is the time left for requesting the Risk Coordinator. The clock $t0$ is reset at transition (*noConvoy, wait*). The invariant $\{t0 < t_{total} - t_{reconfig}\}$ guarantees that after requesting the Risk Coordinator there is still enough time left to execute the actual reconfiguration. If $t0$ exceeds the time limit, the system switches back to *noConvoy* with the side effect *onTimeout()*.

2.3 Allowing Required Reconfigurations

In some cases blocking transitions is not acceptable, e.g. reconfiguration in case of a component failure, or the request to the Risk Coordinator exceeds the time limits. Consequently, not all reconfiguring transitions can be Safety Transitions.

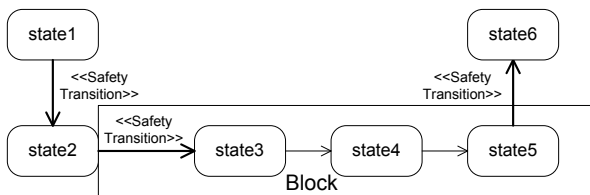


Figure 4: Safe Reconfiguration and Required Behavior

In order to still guarantee safe reconfigurations, we block transitions to configurations from which unsafe configurations are reachable via non blockable transitions. Figure 4 shows an exemplary path containing non blockable transitions, which are drawn as thin lines. If the configuration in *state5* was not allowed and we had to block transition (*state4, state5*) as a consequence, we also would have to block transitions (*state2, state3*) and (*state3, state4*) (marked by a grey shadow). This check has to be applied to reconfigurations of the other system components as well, as a reconfiguration of one component can lead to a reconfiguration of another component.

Instead of sending a request to the Risk Coordinator when executing a non blockable transition, the component informs the risk coordinator of the executed reconfiguration. Since the non blockable transition must not be delayed, this information is sent immediately. Consequently, it is possible that the Risk Coordinator processes the update later than the reconfiguration takes place and doesn't know about the real system state. However, this is no threat to the safety as we already excluded unsafe configurations from the last

blockable transition preceding the non blockable (cf. Figure 4).

2.4 Computing the Risk

The Risk Coordinator computes the risk of a configuration in order to decide whether the target configuration satisfies the maximum risk level or not. The risk is computed by multiplying the probability of the hazard in the target configuration with the damage of an accident that could result from the hazard. Currently, the hazard probabilities of the different system configurations are computed offline with the approach of [GT06] and stored in the Risk Coordinator. The damage is adapted continuously, e.g. when a RailCab is empty or contains hazardous materials.

2.5 Online Hazard Analysis

The approach presented so far only allows for the safe reconfiguration of a system of which all configurations are known before runtime as the hazard probabilities of all these configurations are pre-computed. For example, it is not possible to model infinitely long RailCab convoys as proposed in [THHO08]. In the following, we sketch an approach for computing the hazard probabilities during runtime.

Since the current configuration of the total system is only known during runtime, the Risk Coordinator has to build the failure propagation model of the total system. The component's failure propagation model is stored in each component's Sub Coordinator. The Sub Coordinator transmits this failure propagation to the Risk Coordinator that computes the global failure model and performs a hazard analysis on the complete system. Once the hazard probabilities are known, the risk can be determined and the requested reconfiguration can be approved or rejected.

3. TOOL SUPPORT

The approach presented in this work has been implemented and embedded into several Fujaba4Eclipse plugins [THMvD08, BGH⁺07]. These plugins support the modeling and analysis of safety-critical embedded systems with reconfiguration.

In this work we extended hybrid Reconfiguration Charts by Safety Transitions as presented in Section 2.2.

We implemented a simulation interface for the Risk Coordinator component that supports checking whether configurations satisfy a given risk level. Figure 5 depicts the *Simulation View* of the plugin. The editor shows a component structure (configuration) consisting of three RailCabs. Below each system component is associated a hazard probability for a specific hazard. The total system is assigned a damage. Further, we can specify the maximum allowed risk for the total system and read the current risk. In order to simulate a reconfiguration, we select a component and choose a target configuration ("convoy") from the drop down menu. After pressing the Reconfigure button, the risk of the target configuration is computed and compared to the maximum allowed risk.

We are currently working on implementing the Risk Coordinator component as well as the online Hazard Analysis

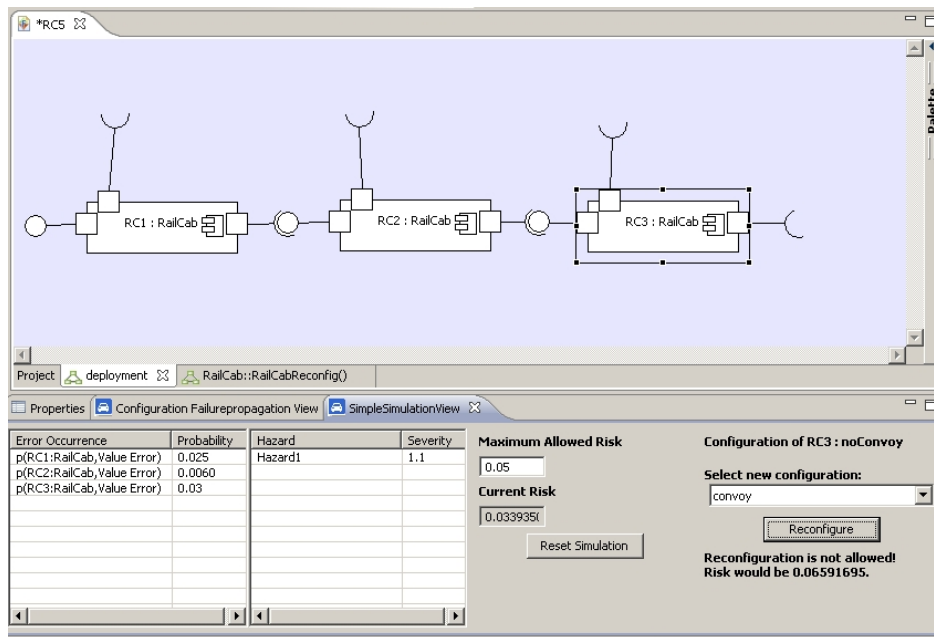


Figure 5: Simulation Interface for the Risk Coordinator

as sketched in Section 2.5 in order to fully integrate the presented approach into the Fujaba4Eclipse Real-Time Tool Suite.

4. CONCLUSION AND FUTURE WORK

We presented a risk analysis approach and its implementation in the Fujaba4Eclipse Real-Time Tool Suite which addresses self-* systems and dynamically changing damage values. The approach is twofold. First, the architecture is extended by a Risk Coordinator which checks whether a re-configuration is safe w.r.t. the current damage. Second, the modelling language for the specification of the re-configuration is extended by safety transitions which encapsulate the communication of each individual component with the Risk Coordinator.

The Risk Coordinator is a single point of failure in the current architectural approach. As we already consider self-* systems, we may add self-healing capabilities to the system as e.g. in [TG04] to fully operate even in case of failures.

It has to be noted that our approach requires quantitative data for both the hazard probability as well as the associated damage in order to compute the risk and to decide whether a configuration is allowed or not. Leveson [Lev95] says that quantitative data should be used with extreme care. Thus, we are currently investigating whether our approach can be extended to qualitative reasoning using probability and damage classes.

Our current approach does only block reconfigurations from a safe to an unsafe configuration. Due to changes to the damage during runtime a safe configuration may become unsafe. Therefore, we currently work on complementing our approach by forcing a reconfiguration from an unsafe to a safe configuration.

5. REFERENCES

- [BGH⁺07] Sven Burmester, Holger Giese, Stefan Henkler, Martin Hirsch, Matthias Tichy, Alfonso Gambuzza, Eckehard MÜch, and Henner Vöcking. Tool support for developing advanced mechatronic systems: Integrating the fujaba real-time tool suite with camel-view. In *Proc. of the 29th International Conference on Software Engineering (ICSE)*, Minneapolis, Minnesota, USA, pages 801–804. IEEE Computer Society Press, May 2007.
- [GT06] Holger Giese and Matthias Tichy. Component-based hazard analysis: Optimal designs, product lines, and online-reconfiguration. In *Proc. of the 25th International Conference on Computer Safety, Security and Reliability (SAFECOMP)*, Gdansk, Poland, Lecture Notes in Computer Science (LNCS), pages 156–169. Springer Verlag, September 2006.
- [GTS04] Holger Giese, Matthias Tichy, and Daniela Schilling. Compositional Hazard Analysis of UML Components and Deployment Models. In *Proc. of the 23rd International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, Potsdam, Germany, volume 3219 of *Lecture Notes in Computer Science*. Springer Verlag, September 2004.
- [Lap92] Jean Claude Laprie, editor. *Dependability : basic concepts and terminology in English, French, German, Italian and Japanese [IFIP WG 10.4, Dependable Computing and Fault Tolerance]*, volume 5 of *Dependable computing and fault tolerant systems*. Springer Verlag, Wien, 1992.
- [Lev95] Nancy G. Leveson. *Safeware: System Safety*

- and Computers*. Addison-Wesley, 1995.
- [Sto96] Neil Storey. *Safety-Critical Computer Systems*. Addison-Wesley, 1996.
- [TG04] Matthias Tichy and Holger Giese. A self-optimizing run-time architecture for configurable dependability of services. In Rogério de Lemos, Cristina Gacek, and Alexander Romanovsky, editors, *Architecting Dependable Systems II*, volume 3069 of *Lecture Notes in Computer Science (LNCS)*, pages 25–51. Springer Verlag, 2004.
- [THHO08] Matthias Tichy, Stefan Henkler, Jörg Holtmann, and Simon Oberthür. Towards a Transformation Language for Component Structures. In *Postproc. of the 4th Workshop on Object-oriented Modeling of Embedded Real-Time Systems (OMER 4)*, Paderborn, Germany, pages 27–39, 2008.
- [THMvD08] Matthias Tichy, Stefan Henkler, Matthias Meyer, and Markus von Detten. Safety of component-based systems: Analysis and improvement using fujaba4eclipse. In *Companion Proceedings of the 30th International Conference on Software Engineering (ICSE)*, Leipzig, Germany, pages 1–2, May 2008.
- [YA02] Sherif M. Yacoub and Hany H. Ammar. A methodology for architecture-level reliability risk analysis. *IEEE Trans. Softw. Eng.*, 28(6):529–547, 2002.